“一切皆文件” linux中的至理名言。对于linux网络子系统，依然适用。sockfs就是最直接的证明。接下来我们分析linux网络子系统的基石--sockfs。后续我们会基于sockfs来对linux网络子系统做全面深入的分析。

        分析任何东西，都要先找到他的入口。net/socket.c--sock\_init()就是我们需要分析的网络子系统的入口。

            static int \_\_init sock\_init(void)

            {

                    skb\_init(); //初始化skbuff的slab cache  名为skbuff\_head\_cache  skb\_init 定义与net/core/skbuff.c中

                                        //对slab cache不了解的同学可以参考下深入linux内核架构中内存管理的相关章节

                    init\_inodecache();// 初始化socket\_alloc类型的slab cache cache名为sock\_inode\_cache

                                                // 返回地址存储在全局变量sock\_inode\_cachep指针中

                    err = register\_filesystem(&sock\_fs\_type); //注册sockfs文件系统 注册原理同其他文件系统注册一直，都是将

                                                                                        //对应的file\_system\_type 变量插入全局变量file\_systems指向的链表中

                    sock\_mnt = kern\_mount(&sock\_fs\_type); //执行文件系统挂在操作，接下来重点讨论

                    ptp\_classifier\_init();//后续再分析

            }

            以上是sockfs初始化并挂载的关键步骤，省略部分代码。也是网络子系统的初始化。在sockfs注册过程用到了sock\_fs\_type变量。那么我们看下其定义及其结构的定义：

            net/socket.c

            static struct file\_system\_type sock\_fs\_type = {

                     .name = "sockfs",

                     .mount = sockfs\_mount,

                     .kill\_sb = kill\_anon\_super,

            };

            struct file\_system\_type {

                     const char \*name;

                     int fs\_flags;

                     #define FS\_REQUIRES\_DEV 1

                     #define FS\_BINARY\_MOUNTDATA 2

                     #define FS\_HAS\_SUBTYPE 4

                     #define FS\_USERNS\_MOUNT 8 /\* Can be mounted by userns root \*/

                     #define FS\_RENAME\_DOES\_D\_MOVE 32768 /\* FS will handle d\_move() during rename() internally. \*/

                     struct dentry \*(\*mount) (struct file\_system\_type \*, int, const char \*, void \*);

                     void (\*kill\_sb) (struct super\_block \*);

                     struct module \*owner;

                     struct file\_system\_type \* next;

                     struct hlist\_head fs\_supers;

            }

            注册过程比较简单，就是向file\_systems链表中插入file\_system\_type变量。接下来分析 kern\_mount挂载过程。

            include/linux/fs.h

            #define kern\_mount(type) kern\_mount\_data(type, NULL) //宏定义

            fs/namespace.c

            最终实现：

            struct vfsmount \*kern\_mount\_data(struct file\_system\_type \*type, void \*data)

            {

                 struct vfsmount \*mnt;

                 mnt = vfs\_kern\_mount(type, MS\_KERNMOUNT, type->name, data);

                 if (!IS\_ERR(mnt)) {

                 /\*

                   \* it is a longterm mount, don't release mnt until

                   \* we unmount before file sys is unregistered

                   \*/

                      real\_mount(mnt)->mnt\_ns = MNT\_NS\_INTERNAL;

                 }

                 return mnt;

            }

            调用 vfs\_kern\_mount()

            在fs/namespace.c文件中定义

            实现：

               struct vfsmount \*

               vfs\_kern\_mount(struct file\_system\_type \*type, int flags, const char \*name, void \*data)

               {

                        struct mount \*mnt;

                        struct dentry \*root;

                        if (!type)

                             return ERR\_PTR(-ENODEV);

                        mnt = alloc\_vfsmnt(name); //分配mount结构，并初始化

                        if (!mnt)

                            return ERR\_PTR(-ENOMEM);

                        if (flags & MS\_KERNMOUNT)

                             mnt->mnt.mnt\_flags = MNT\_INTERNAL; //设置内部挂载标志

                        root = mount\_fs(type, flags, name, data);

                        if (IS\_ERR(root)) {

                                mnt\_free\_id(mnt);

                                free\_vfsmnt(mnt);

                                return ERR\_CAST(root);

                          }

                        mnt->mnt.mnt\_root = root;

                        mnt->mnt.mnt\_sb = root->d\_sb;

                        mnt->mnt\_mountpoint = mnt->mnt.mnt\_root;

                        mnt->mnt\_parent = mnt;

                        lock\_mount\_hash();

                        list\_add\_tail(&mnt->mnt\_instance, &root->d\_sb->s\_mounts);

                        unlock\_mount\_hash();

                        return &mnt->mnt;

               }

                这一部分代码比较简单，就是分配mount(挂载点)结构，同时调用mount\_fs执行挂载操作，如下

                // fs/super.c中定义

                struct dentry \*

                mount\_fs(struct file\_system\_type \*type, int flags, const char \*name, void \*data)

                {

                         struct dentry \*root;

                         struct super\_block \*sb;

                         char \*secdata = NULL;

                         int error = -ENOMEM;

                         if (data && !(type->fs\_flags & FS\_BINARY\_MOUNTDATA)) {

                              secdata = alloc\_secdata();

                              if (!secdata)

                               goto out;

                              error = security\_sb\_copy\_data(data, secdata);

                              if (error)

                                   goto out\_free\_secdata;

                         }

                         root = type->mount(type, flags, name, data);// 调用sock\_fs\_type成员mount 即sockfs\_mount（）函数

                         if (IS\_ERR(root)) {

                              error = PTR\_ERR(root);

                              goto out\_free\_secdata;

                         }

                         sb = root->d\_sb;

                         BUG\_ON(!sb);

                         WARN\_ON(!sb->s\_bdi);

                         sb->s\_flags |= MS\_BORN;

                         error = security\_sb\_kern\_mount(sb, flags, secdata);

                         if (error)

                         goto out\_sb;

 /\*

  \* filesystems should never set s\_maxbytes larger than MAX\_LFS\_FILESIZE

  \* but s\_maxbytes was an unsigned long long for many releases. Throw

  \* this warning for a little while to try and catch filesystems that

  \* violate this rule.

  \*/

                         WARN((sb->s\_maxbytes < 0), "%s set sb->s\_maxbytes to "

                              "negative value (%lld)\n", type->name, sb->s\_maxbytes);

                         up\_write(&sb->s\_umount);

                         free\_secdata(secdata);

                         return root;

                     out\_sb:

                         dput(root);

                         deactivate\_locked\_super(sb);

                     out\_free\_secdata:

                         free\_secdata(secdata);

                     out:

                             return ERR\_PTR(error);

                }

            该函数调用type->mount() 执行挂载操作，返回sockfs根目录root。type->mount()指向sockfs\_mount()函数。如下：

            net/socket.c

            static struct dentry \*sockfs\_mount(struct file\_system\_type \*fs\_type,

                    int flags, const char \*dev\_name, void \*data)

            {

                 return mount\_pseudo\_xattr(fs\_type, "socket:", &sockfs\_ops,

                          sockfs\_xattr\_handlers,

                          &sockfs\_dentry\_operations, SOCKFS\_MAGIC);

            }该函数简单封装mount\_pseudo\_xattr()函数，该函数主要创建super\_block同时创建根节点inode，dentry等相关关系。完成sockfs的初始化

            最终sockfs形成如下框架：

